OpenCV

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

About the Tutorial

OpenCV is a cross-platform library using which we can develop real-time **computer vision applications**. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection. In this tutorial, we explain how you can use OpenCV in your applications.

Audience

This tutorial has been prepared for beginners to make them understand the basics of OpenCV library. We have used the Java programming language in all the examples, therefore you should have a basic exposure to Java in order to benefit from this tutorial.

Prerequisites

For this tutorial, it is assumed that the readers have a prior knowledge of Java programming language. In some of the programs of this tutorial, we have used JavaFX for GUI purpose. So, it is recommended that you go through our JavaFX tutorial before proceeding further - http://www.tutorialspoint.com/javafx/.

Copyright & Disclaimer

Table of Contents

1. OpenCV – Overview

OpenCV is a cross-platform library using which we can develop real-time **computer vision** applications. It mainly focuses on image processing, video capture and analysis including features like face detection and object detection.

Let's start the chapter by defining the term "Computer Vision".

Computer Vision

Computer Vision can be defined as a discipline that explains how to reconstruct, interrupt, and understand a 3D scene from its 2D images, in terms of the properties of the structure present in the scene. It deals with modeling and replicating human vision using computer software and hardware.

Computer Vision overlaps significantly with the following fields:

- **Image Processing:** It focuses on image manipulation.
- **Pattern Recognition:** It explains various techniques to classify patterns.
- **Photogrammetry:** It is concerned with obtaining accurate measurements from images.

Computer Vision Vs Image Processing

Image processing deals with image-to-image transformation. The input and output of image processing are both images.

Computer vision is the construction of explicit, meaningful descriptions of physical objects from their image. The output of computer vision is a description or an interpretation of structures in 3D scene.

Applications of Computer Vision

Here we have listed down some of major domains where Computer Vision is heavily used.

Robotics Application

- Localization — Determine robot location automatically
- Navigation
- Obstacles avoidance
- Assembly (peg-in-hole, welding, painting)
- Manipulation (e.g. PUMA robot manipulator)
- Human Robot Interaction (HRI): Intelligent robotics to interact with and serve people

Medicine Application

- Classification and detection (e.g. lesion or cells classification and tumor detection)

- 2D/3D segmentation
- 3D human organ reconstruction (MRI or ultrasound)
- Vision-guided robotics surgery

Industrial Automation Application

- Industrial inspection (defect detection)
- Assembly
- Barcode and package label reading
- Object sorting
- Document understanding (e.g. OCR)

Security Application

- Biometrics (iris, finger print, face recognition)
- Surveillance – Detecting certain suspicious activities or behaviors

Transportation Application

- Autonomous vehicle
- Safety, e.g., driver vigilance monitoring

Features of OpenCV Library

Using OpenCV library, you can –

- Read and write images
- Capture and save videos
- Process images (filter, transform)
- Perform feature detection
- Detect specific objects such as faces, eyes, cars, in the videos or images.
- Analyze the video, i.e., estimate the motion in it, subtract the background, and track objects in it.

OpenCV was originally developed in C++. In addition to it, Python and Java bindings were provided. OpenCV runs on various Operating Systems such as windows, Linux, OSx, FreeBSD, Net BSD, Open BSD, etc.

This tutorial explains the concepts of OpenCV with examples using Java bindings.

OpenCV Library Modules

Following are the main library modules of the OpenCV library.

Core Functionality

This module covers the basic data structures such as Scalar, Point, Range, etc., that are used to build OpenCV applications. In addition to these, it also includes the multidimensional array **Mat,** which is used to store the images. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.core**.

Image Processing

This module covers various image processing operations such as image filtering, geometrical image transformations, color space conversion, histograms, etc. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.imgproc**.

Video

This module covers the video analysis concepts such as motion estimation, background subtraction, and object tracking. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.video.**

Video I/O

This module explains the video capturing and video codecs using OpenCV library. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.videoio**.

calib3d

This module includes algorithms regarding basic multiple-view geometry algorithms, single and stereo camera calibration, object pose estimation, stereo correspondence and elements of 3D reconstruction. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.calib3d**.

features2d

This module includes the concepts of feature detection and description. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.features2d**.

Objdetect

This module includes the detection of objects and instances of the predefined classes such as faces, eyes, mugs, people, cars, etc. In the Java library of OpenCV, this module is included as a package with the name **org.opencv.objdetect**.

Highgui

This is an easy-to-use interface with simple UI capabilities. In the Java library of OpenCV, the features of this module is included in two different packages namely, **org.opencv.imgcodecs** and **org.opencv.videoio.**

A Brief History of OpenCV

OpenCV was initially an Intel research initiative to advise CPU-intensive applications. It was officially launched in 1999.

- In the year 2006, its first major version, OpenCV 1.0 was released.
- In October 2009, the second major version, OpenCV 2 was released.
- In August 2012, OpenCV was taken by a nonprofit organization OpenCV.org.

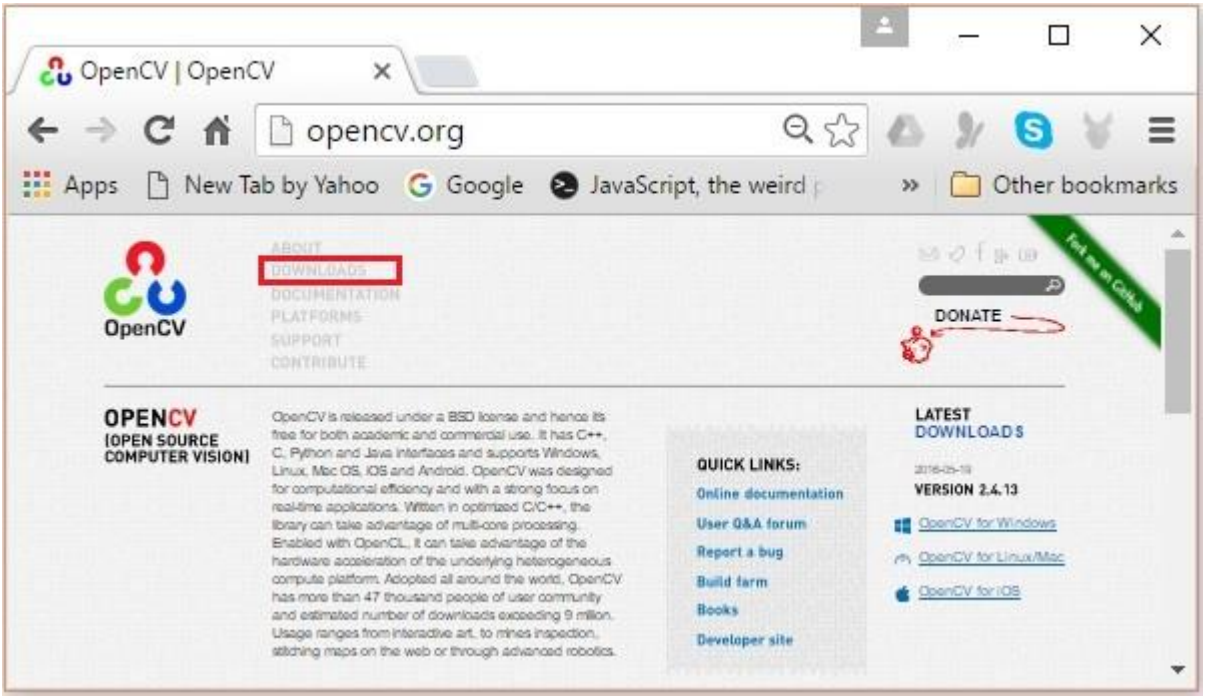
2. OpenCV – Environment

In this chapter, you will learn how to install OpenCV and set up its environment in your system.

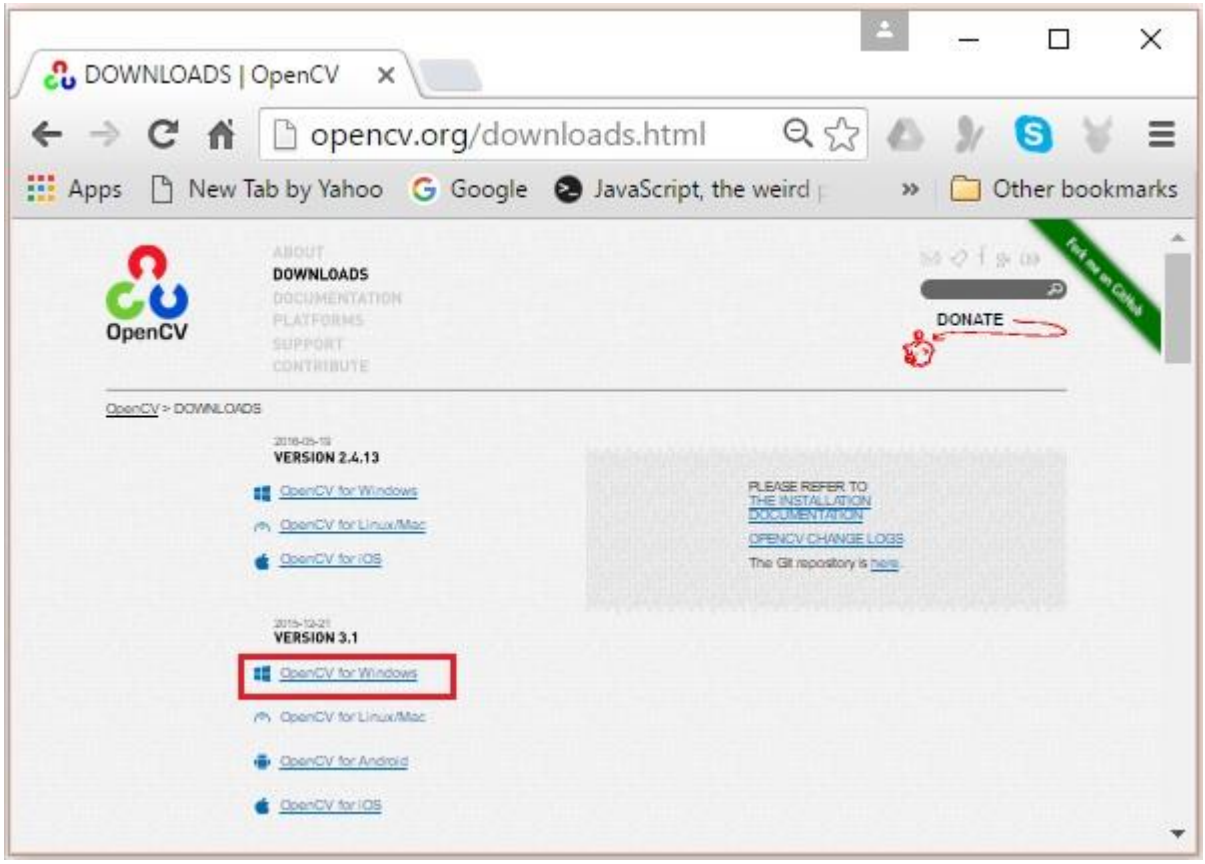
Installing OpenCV

First of all, you need to download OpenCV onto your system. Follow the steps given below.

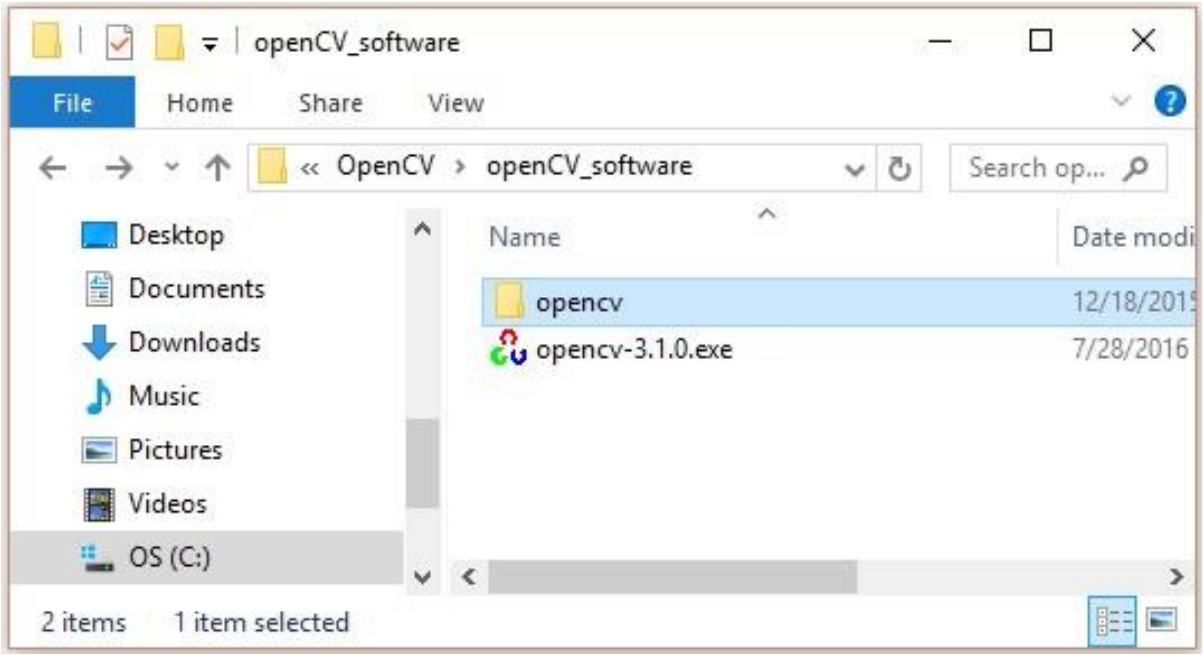
Step 1: Open the homepage of **OpenCV** by clicking the following link: http://opencv.org/ On clicking, you will see its homepage as shown below.



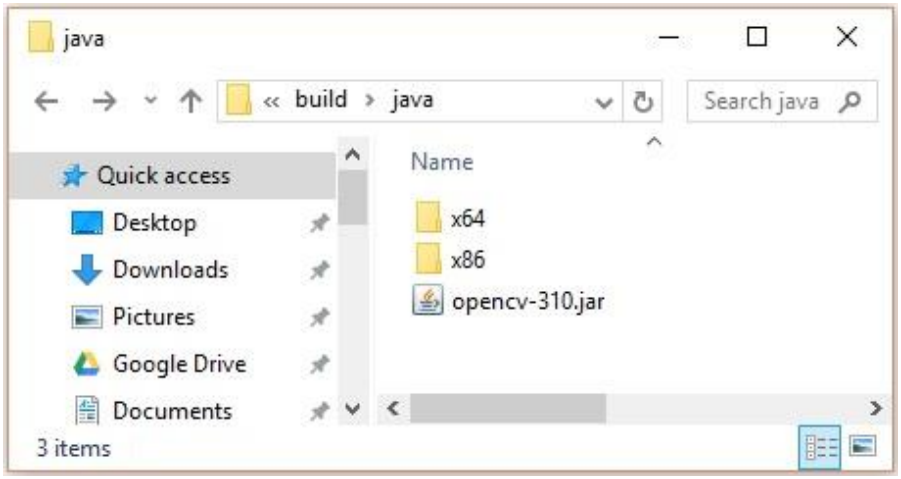
Step 2: Now, click the **Downloads** link highlighted in the above screenshot**.** On clicking, you will be directed to the downloads page of OpenCV.



Step 3: On clicking the highlighted link in the above screenshot, a file named **opencv-3.1.0.exe** will be downloaded. Extract this file to generate a folder **opencv** in your system, as shown in the following screenshot.



Step 4: Open the folder **OpenCV** -> **build** -> **java.** Here you will find the jar file of OpenCV named **opencv-310.jar**. Save this file in a separate folder for further use.



Eclipse Installation

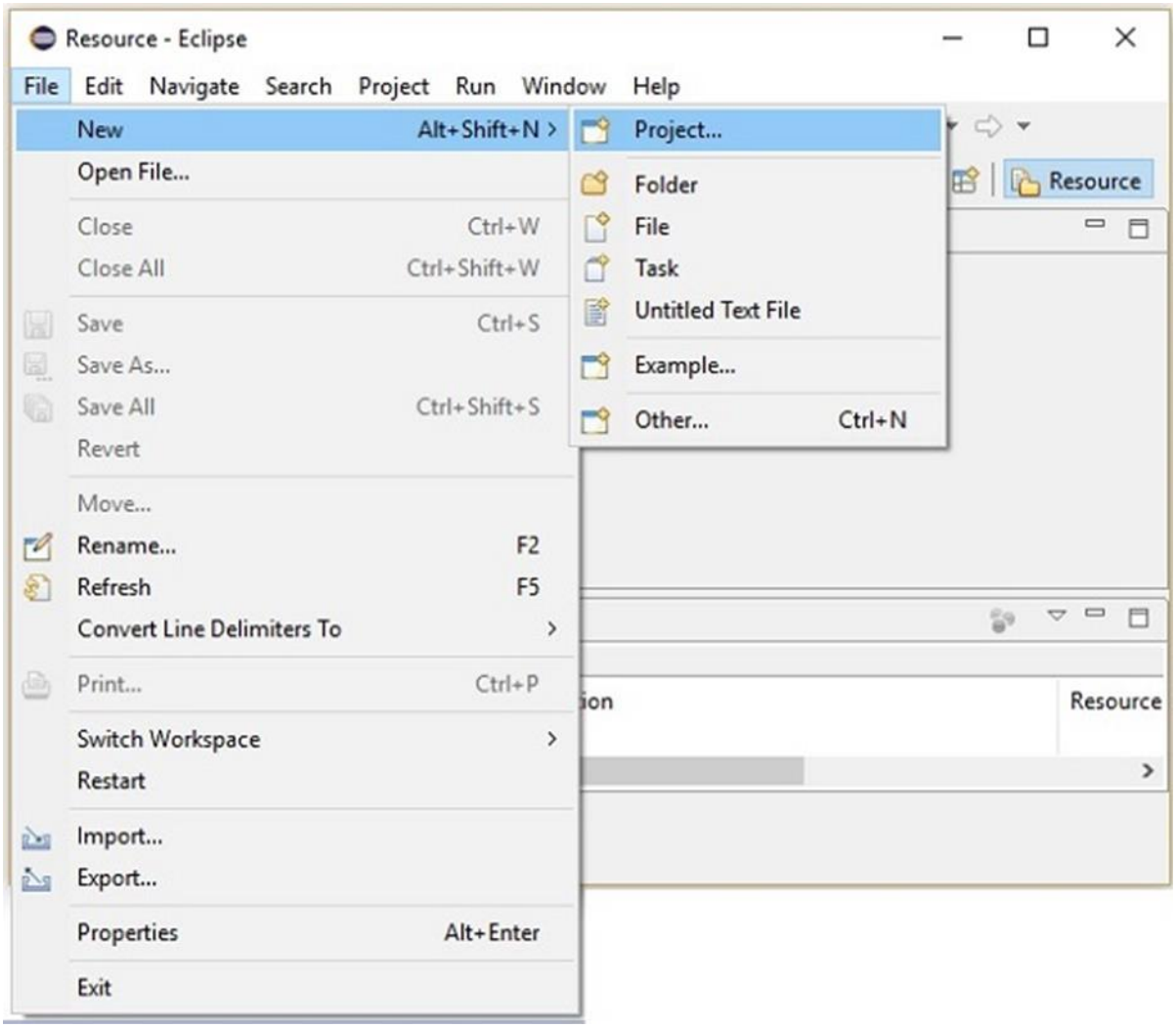
After downloading the required JAR files, you have to embed these JAR files to your Eclipse environment. You can do this by setting the Build Path to these JAR files and by using **pom.xml.**

Setting Build Path

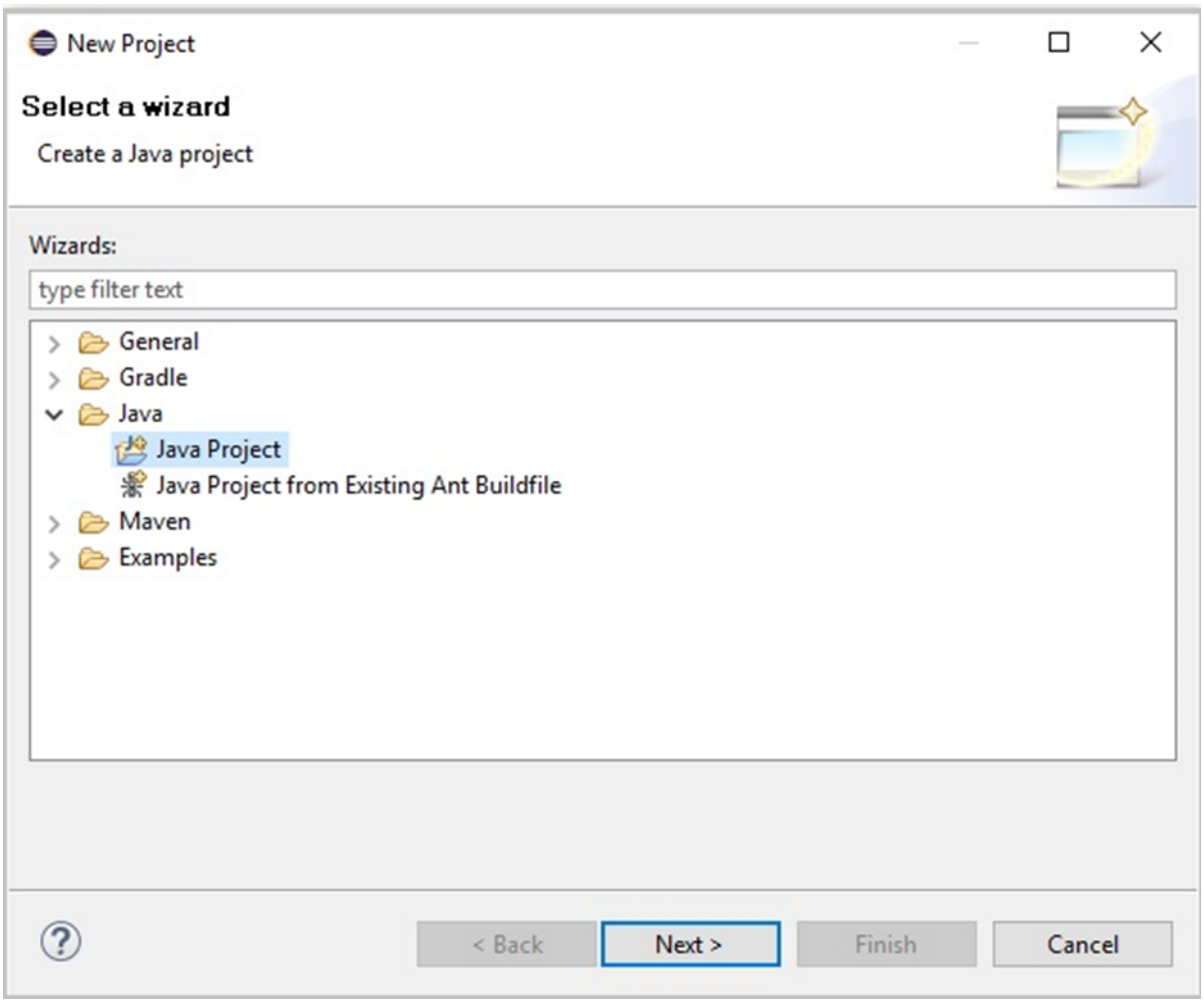
Following are the steps to set up OpenCV in Eclipse:

Step 1: Ensure that you have installed Eclipse in your system. If not, download and install Eclipse in your system.

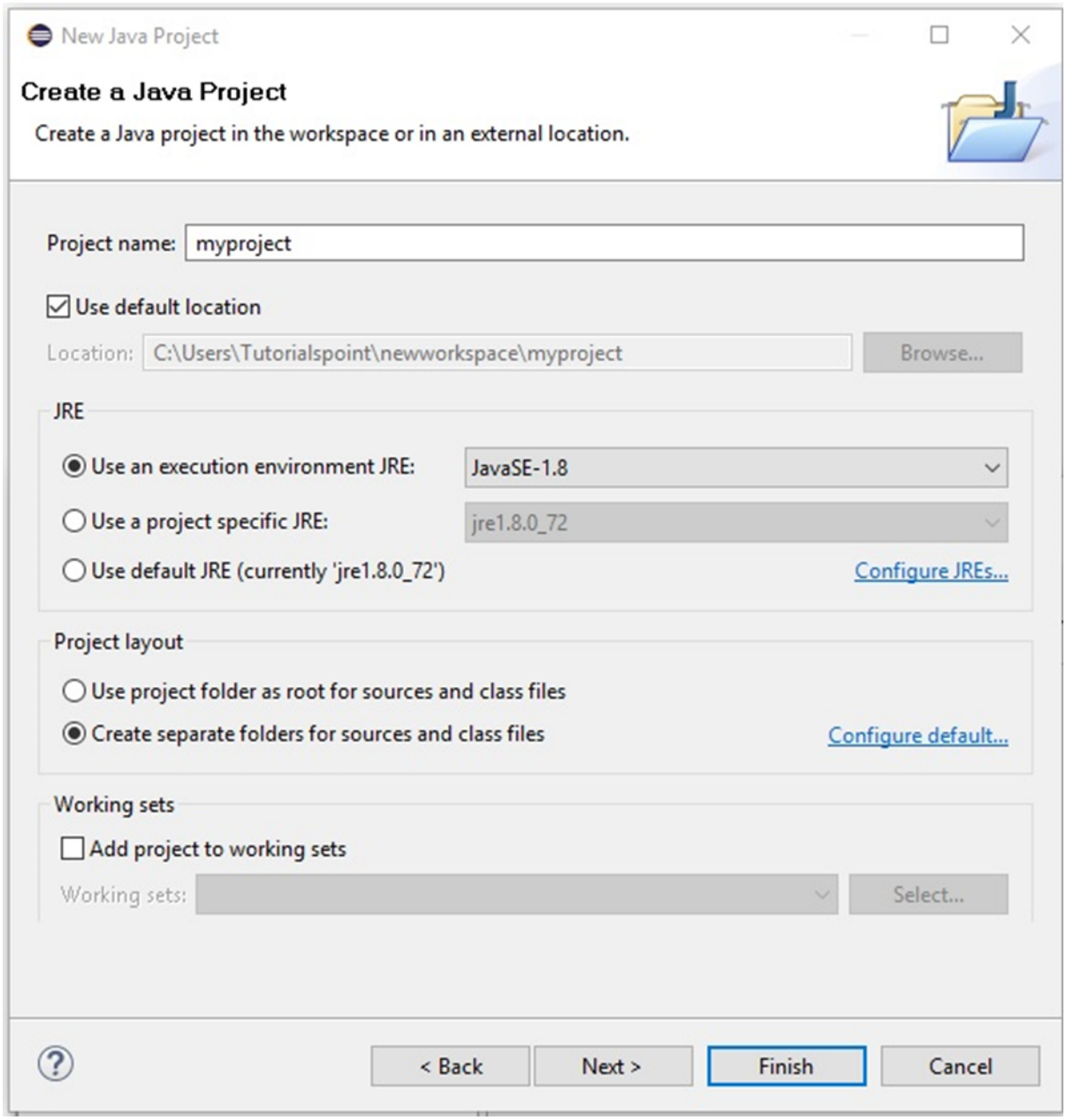
Step 2: Open Eclipse, click on File, New, and Open a new project as shown in the following screenshot.



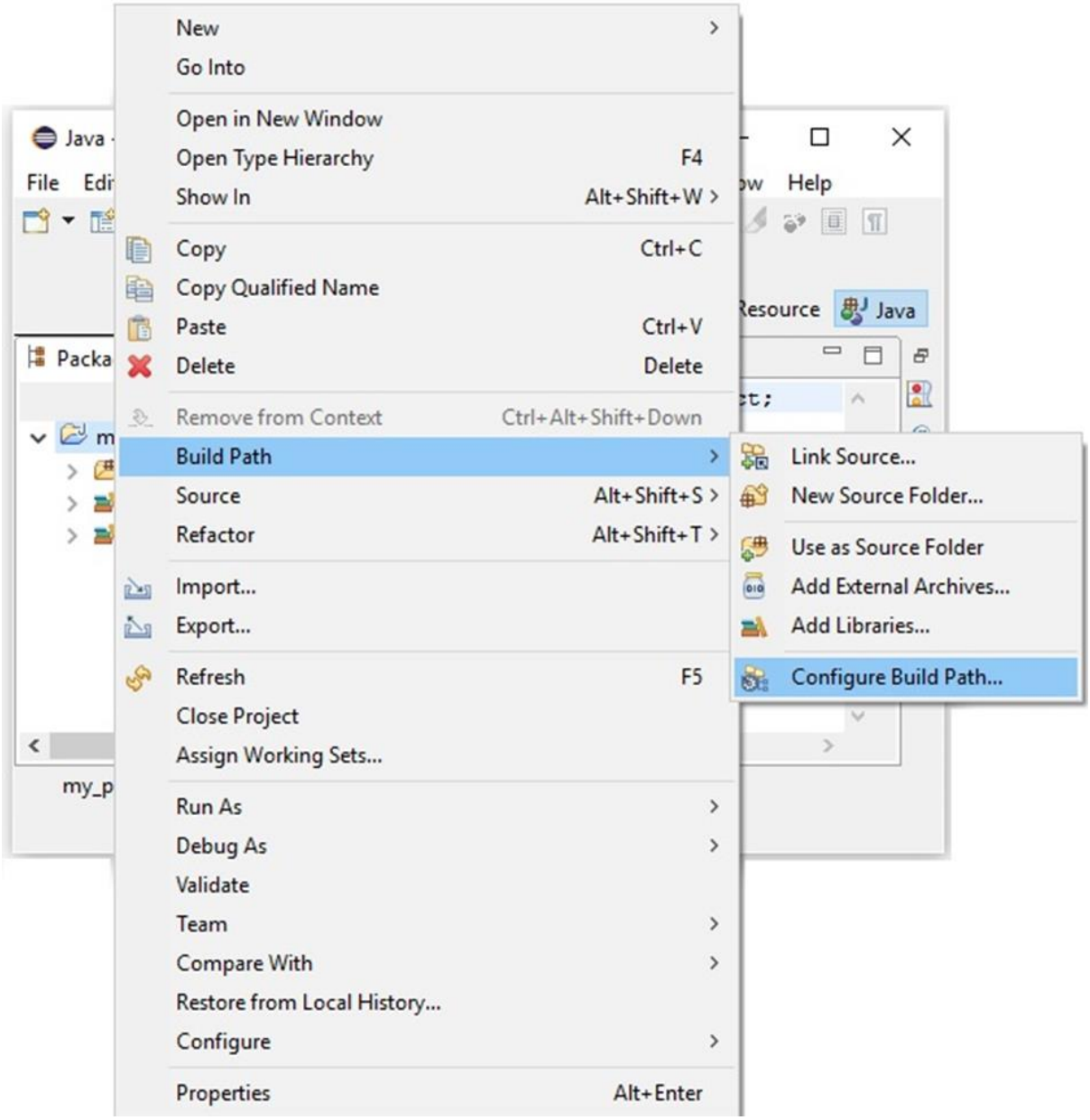
Step 3: On selecting the project, you will get the **New Project** wizard. In this wizard, select Java project and proceed by clicking the **Next** button, as shown in the following screenshot.



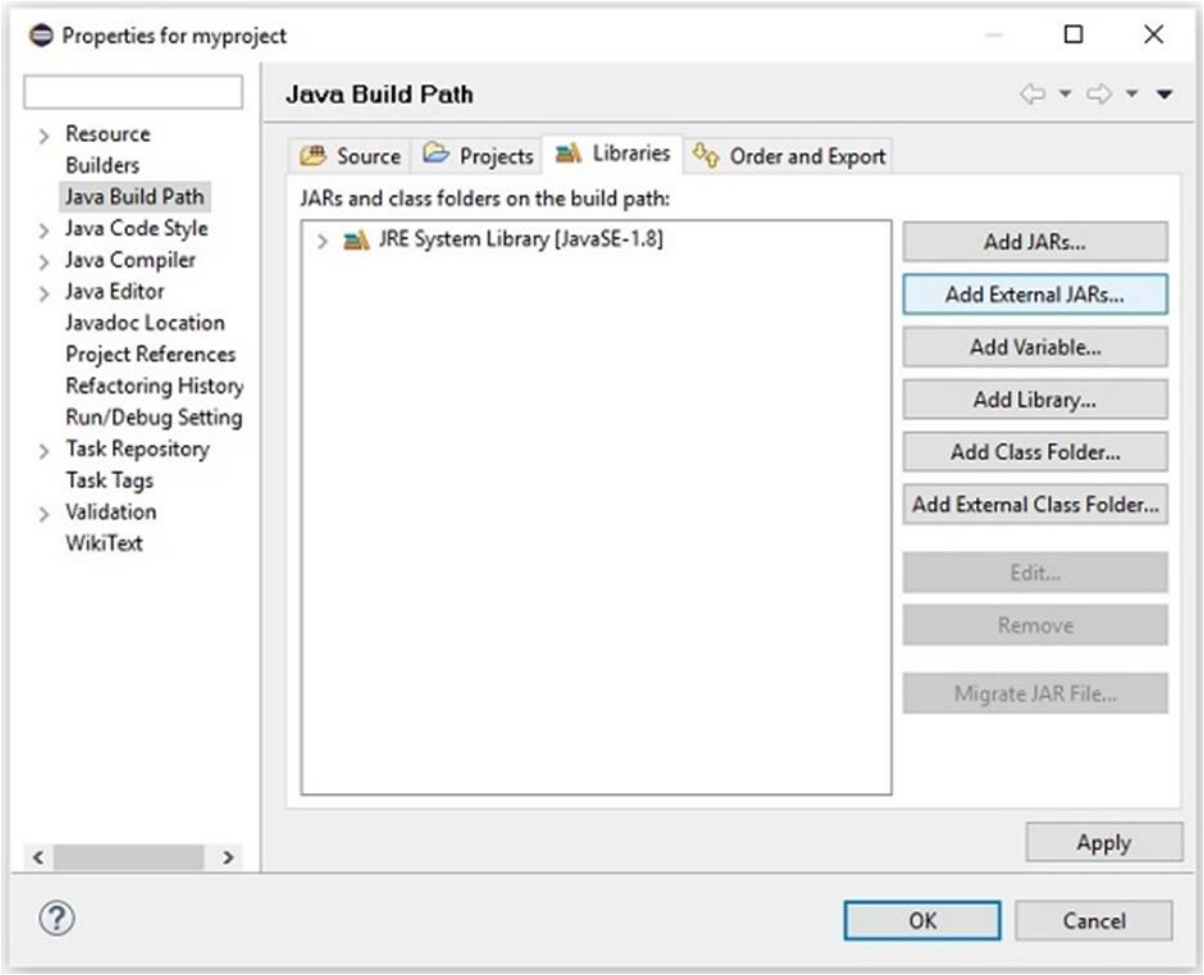
Step 4: On proceeding forward, you will be directed to the **New Java Project wizard**. Create a new project and click **Next**, as shown in the following screenshot.



Step 5: After creating a new project, right-click on it. Select **Build Path** and click **Configure Build Path...** as shown in the following screenshot.

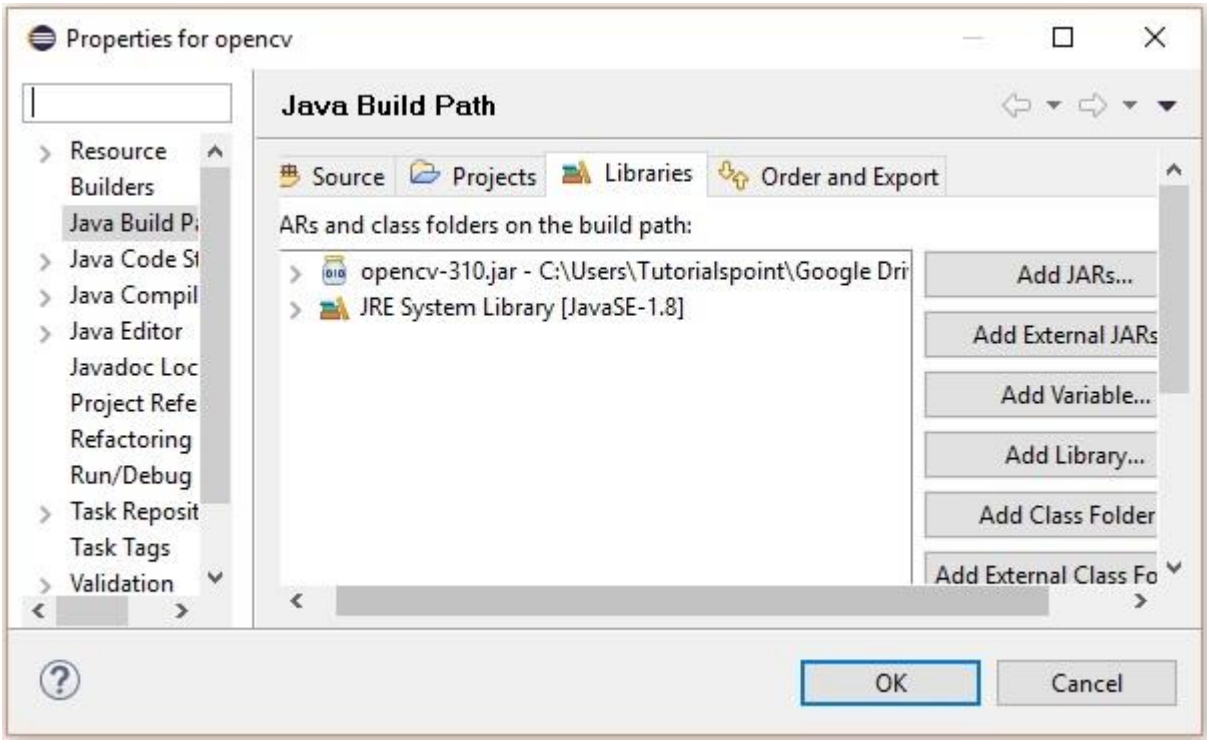


Step 6: On clicking the **Build Path** option, you will be directed to the **Java Build Path wizard**. Click the **Add External JARs** button, as shown in the following screenshot.

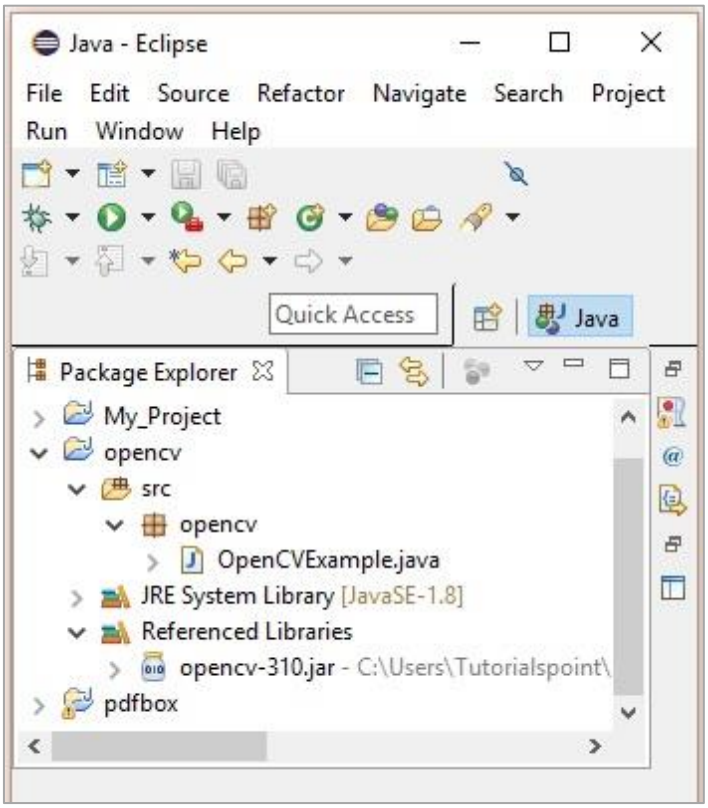


Step 7: Select the path where you have saved the file **opencv-310.jar**.

Step 8: On clicking the **Open** button in the above screenshot, those files will be added to your library.



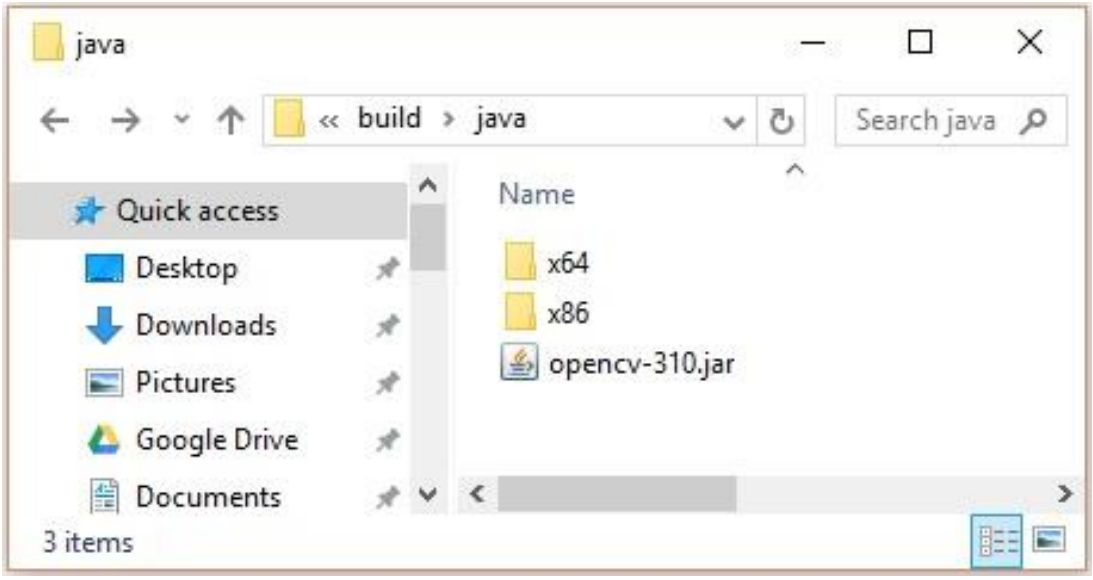
Step 9: On clicking **OK**, you will successfully add the required JAR files to the current project and you can verify these added libraries by expanding the Referenced Libraries.



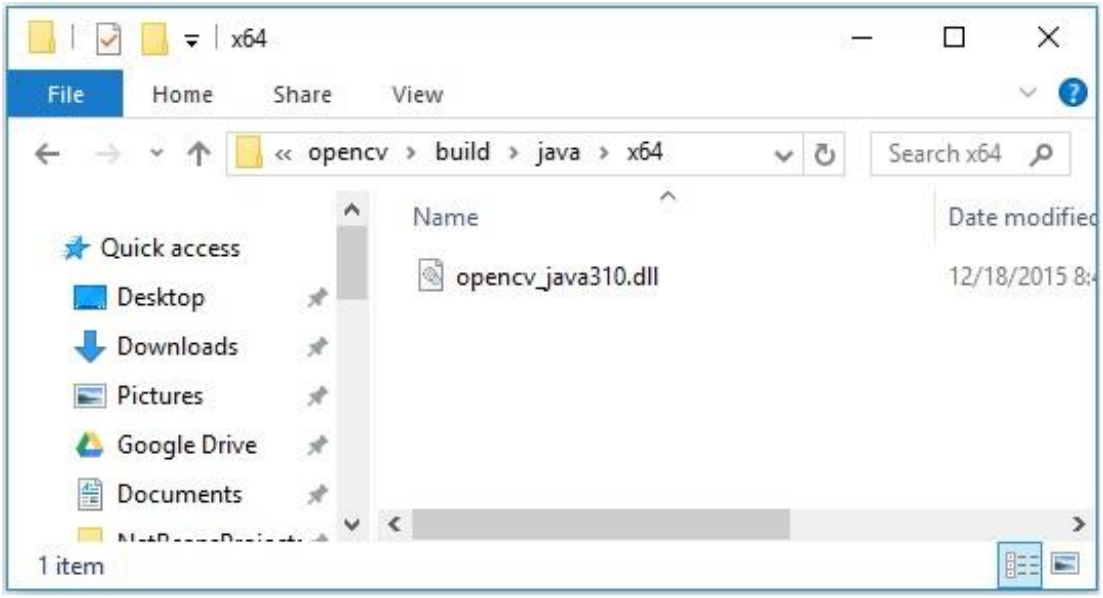
Setting the Path for Native Libraries

In addition to the JAR files, you need to set path for the native libraries (DLL files) of OpenCV.

Location of DLL files: Open the installation folder of **OpenCV** and go to the sub-folder **build -> java.** Here you will find the two folders **x64** (64 bit) and **x86** (32 bit) which contain the **dll** files of OpenCV.

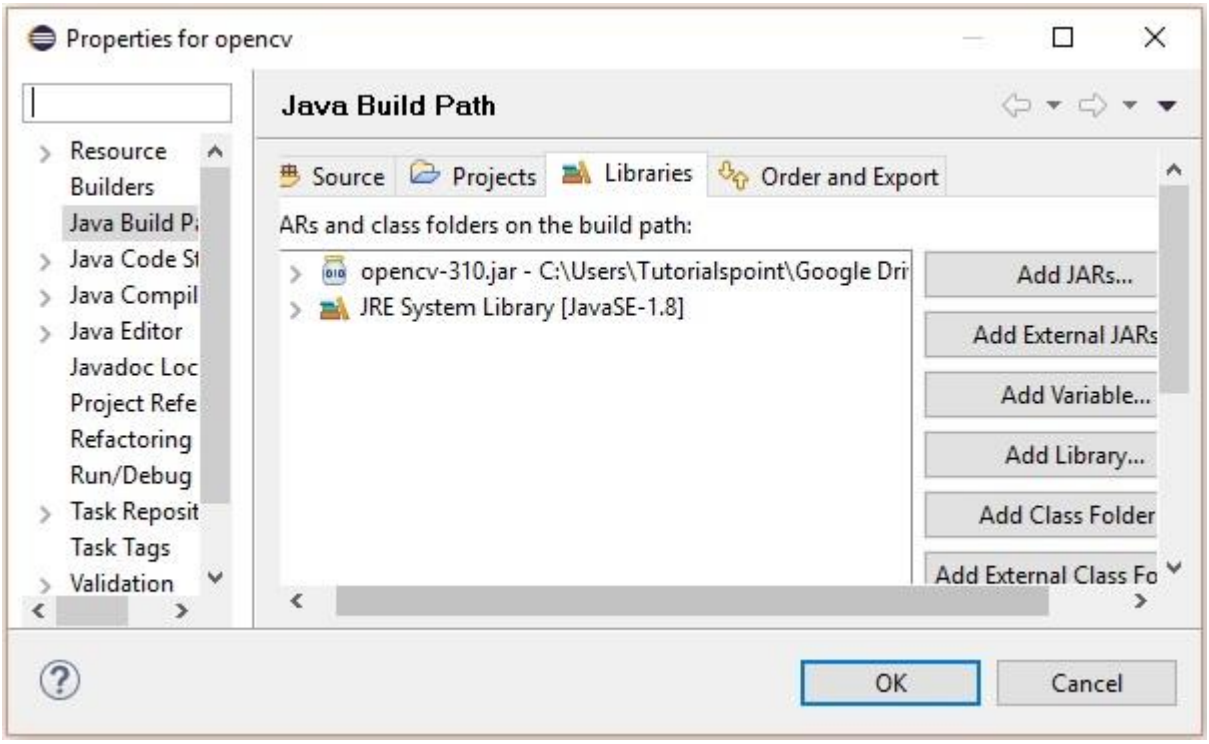


Open the respective folder suitable for your operating system, then you can see the **dll** file, as shown in the following screenshot.

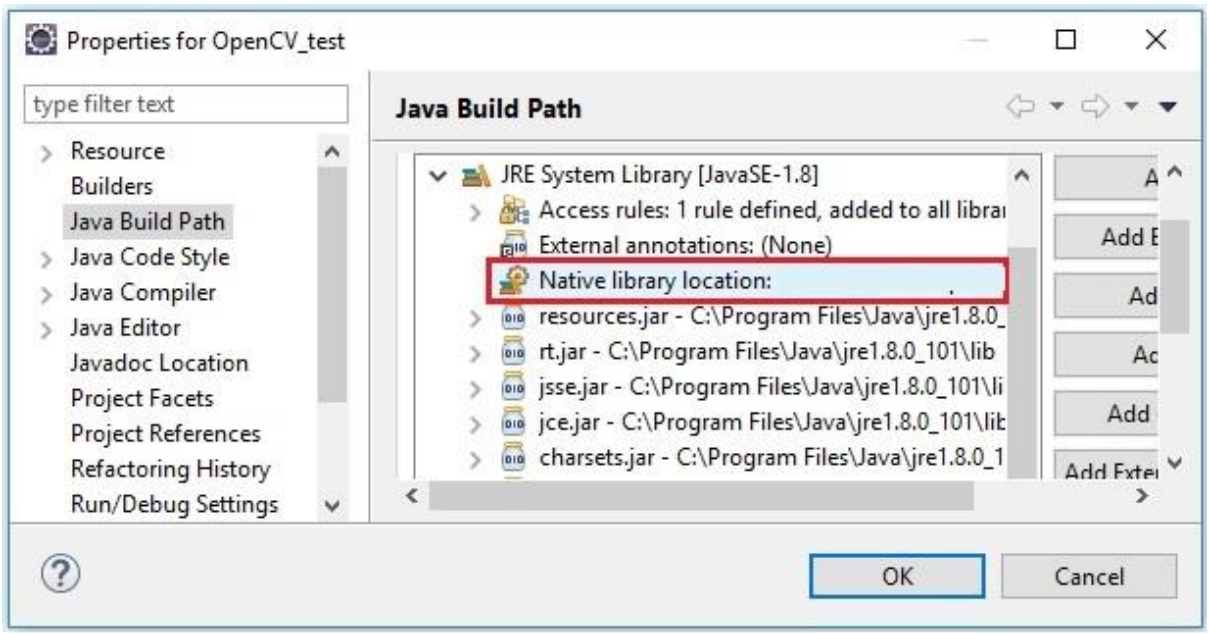


Now, set the path for this file too by following the steps given below—

Step 1: Once again, open the JavaBuildPath window. Here you can observe the added JAR file and the **JRE System Library.**



Step 2: On expanding it, you will get the system libraries and **Native library location**, as highlighted in the following screenshot.



Step 3: Double-click on the **Native library location**. Here, you can see the **Native Library Folder Configuration window** as shown below—

Here, click the button **External Folder...** and select the location of the **dll** file in your system.

3. OpenCV — Storing Images

To capture an image, we use devices like cameras and scanners. These devices record numerical values of the image (Ex: pixel values). OpenCV is a library which processes the digital images, therefore we need to store these images for processing.

The **Mat** class of OpenCV library is used to store the values of an image. It represents an n-dimensional array and is used to store image data of grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms, etc.

This class comprises of two data parts: the **header** and a **pointer**

- **Header:** Contains information like size, method used for storing, and the address of the matrix (constant in size).
- **Pointer:** Stores the pixel values of the image (Keeps on varying).

The Mat Class

The OpenCV Java library provides this class with the same name (**Mat**) within the package **org.opencv.core**.

Constructors

The Mat class of OpenCV Java library has various constructors, using which you can construct the Mat object.

S.No	Constructors and Description
1.	**Mat()** This is the default constructor with no parameters in most cases. We use this to constructor to create an empty matrix and pass this to other OpenCV methods.
2.	**Mat(int rows, int cols, int type)** This constructor accepts three parameters of integer type representing the number of rows and columns in a 2D array and the type of the array (that is to be used to store data).
3.	**Mat(int rows, int cols, int type, Scalar s)** Including the parameters of the previous one, this constructor additionally accepts an object of the class Scalar as parameter.
4.	**Mat(Size size, int type)**

	This constructor accepts two parameters, an object representing the size of the matrix and an integer representing the type of the array used to store the data.
5.	**Mat(Size size, int type, Scalar s)** Including the parameters of the previous one, this constructor additionally accepts an object of the class Scalar as parameter.
6.	**Mat(long addr)**
7.	**Mat(Mat m, Range rowRange)** This constructor accepts an object of another matrix and an object of the class Range representing the range of the rows to be taken to create a new matrix.
8.	**Mat(Mat m, Range rowRange, Range colRange)** Including the parameters of the previous one, this constructor additionally accepts an object of the class. Range representing the column range.
9.	**Mat(Mat m, Rect roi)** This constructor accepts two objects, one representing another matrix and the other representing the **R**egion **O**f **I**nterest.

Note:

- Array type. Use CV_8UC1, ..., CV_64FC4 to create 1-4 channel matrices, or CV_8UC(n), ..., CV_64FC(n) to create multi-channel (up to CV_CN_MAX channels) matrices.
- The type of the matrices were represented by various fields of the class **CvType** which belongs to the package **org.opencv.core.**

Methods and Description

Following are some of the methods provided by the Mat class.

S.No	Methods and Description
1.	**Mat col(int x)** This method accepts an integer parameter representing the index of a column and retrieves and returns that column
2.	**Mat row(int y)**

	This method accepts an integer parameter representing the index of a row and retrieves and returns that row
3.	**int cols()** This method returns the number of columns in the matrix
4.	**int rows()** This method returns the number of rows in the matrix
5.	**Mat setTo(Mat value)** This method accepts an object of the **Mat** type and sets the array elements to the specified value.
6.	**Mat setTo(Scalar s)** This method accepts an object of the **Scalar** type and sets the array elements to the specified value.

Creating and Displaying the Matrix

In this section, we are going to discuss our first OpenCV example. We will see how to create and display a simple OpenCV matrix.

Given below are the steps to be followed to create and display a matrix in OpenCV.

Step 1: Load the OpenCV native library

While writing Java code using OpenCV library, the first step you need to do is to load the native library of OpenCV using the **loadLibrary().** Load the OpenCV native library as shown below.

```
//Loading the core library
System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
```

Step 2: Instantiate the Mat class

Instantiate the Mat class using any of the functions mentioned in this chapter earlier.

```
//Creating a matrix
Mat matrix = new Mat(5, 5, CvType.CV_8UC1, new Scalar(0));
```

Step 3: Fill the matrix using the methods

You can retrieve particular rows/columns of a matrix by passing index values to the methods **row()/col().**

And, you can set values to these using any of the variants of the **setTo()** methods.

```
//Retrieving the row with index 0
Mat row0 = matrix.row(0);

//setting values of all elements in the row with index 0
row0.setTo(new Scalar(1));

//Retrieving the row with index 3
Mat col3 = matrix.col(3);

//setting values of all elements in the row with index 3
col3.setTo(new Scalar(3));
```

Example

You can use the following program code to create and display a simple matrix in Java using OpenCV library.

```
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.CvType;
import org.opencv.core.Scalar;

class DisplayingMatrix{

   public static void main(String[] args) {

      //Loading the core library
      System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

      //Creating a matrix
      Mat matrix = new Mat(5, 5, CvType.CV_8UC1, new Scalar(0));

      //Retrieving the row with index 0
      Mat row0 = matrix.row(0);
```

```
      //setting values of all elements in the row with index 0
      row0.setTo(new Scalar(1));

      //Retrieving the row with index 3
      Mat col3 = matrix.col(3);

      //setting values of all elements in the row with index 3
      col3.setTo(new Scalar(3));

      //Printing the matrix
      System.out.println("OpenCV Mat data:\n" + matrix.dump());
   }
}
```

On executing the above program, you will get the following output.

```
OpenCV Mat data:
[  1,   1,   1,   3,   1;
   0,   0,   0,   3,   0;
   0,   0,   0,   3,   0;
   0,   0,   0,   3,   0;
   0,   0,   0,   3,   0]
```

Loading Image using JavaSE API

The **BufferedImage** class of the **java.awt.image.BufferedImage** package is used to store an image and the **ImageIO** class of the package **import javax.imageio** provides methods to read and write Images.

Example

You can use the following program code to load and save images using JavaSE library.

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class LoadingImage_JSE_library {
   public static void main( String[] args ) throws IOException {
```

```
      //Input File
      File input = new File("C:/EXAMPLES/OpenCV/sample.jpg");

      //Reading the image
      BufferedImage image = ImageIO.read(input);

      //Saving the image with a different name
      File ouptut = new File("C:/OpenCV/sample.jpg");
      ImageIO.write(image, "jpg", ouptut);

      System.out.println("image Saved");
   }
}
```

On executing the above program, you will get the following output.

```
image Saved
```

If you open the specified path, you can observe the saved image as follows—

End of ebook preview

If you liked what you saw…

Buy it from our store @ **https://store.tutorialspoint.com**